ri_block.c - EXHIBIT C

```
/*****************************************************************************
#   Copyright 1997 Silicon Graphics, Inc.  ALL RIGHTS RESERVED.
#
#   UNPUBLISHED -- Rights reserved under the copyright laws of the United
#   States.   Use of a copyright notice is precautionary only and does not
#   imply publication or disclosure.
#
#   THIS SOFTWARE CONTAINS CONFIDENTIAL AND PROPRIETARY INFORMATION OF
#   SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION, DISTRIBUTION, OR
#   DISCLOSURE IS STRICTLY PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN
#   PERMISSION OF SILICON GRAPHICS, INC.
#****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <strings.h>
#include <math.h>
#include "ri.h"
#include "ri_state.h"
#include "xwin.h"
#include <GL/gl.h>
#include <GL/glu.h>

RiOptions ri_options;
RiOptions *CurOptions = &ri_options;
RiAttributes *CurAttributes;
Dlist *CurDlist = NULL;
RiJumpTable JumpIm;
RiJumpTable JumpLC;
RiJumpTable *JumpCur;

/* XXX */
Node *RiScene = NULL;

Shader *globallight = NULL;
Shader *framelight = NULL;
Shader *worldlight = NULL;
int LightNumber = 0;

int RenderState = 0;

extern void delete_lights(Shader *);

static void init_jumptables(void)
{
    JumpIm.AttributeBegin = __riim_AttributeBegin;
    JumpIm.AttributeEnd = __riim_AttributeEnd;
    JumpIm.Basis = __riim_Basis;
    JumpIm.Sides = __riim_Sides;
    JumpIm.Orientation = __riim_Orientation;
    JumpIm.ReverseOrientation = __riim_ReverseOrientation;
    JumpIm.Identity = __riim_Identity;
    JumpIm.Transform = __riim_Transform;
    JumpIm.ConcatTransform = __riim_ConcatTransform;
    JumpIm.Perspective = __riim_Perspective;
    JumpIm.Translate = __riim_Translate;
    JumpIm.Rotate = __riim_Rotate;
    JumpIm.Scale = __riim_Scale;
    JumpIm.TransformBegin = __riim_TransformBegin;
    JumpIm.TransformEnd = __riim_TransformEnd;
    JumpIm.SurfaceV = __riim_SurfaceV;
    JumpIm.Illuminate = __riim_Illuminate;

    JumpLC.AttributeBegin = __rilc_AttributeBegin;
    JumpLC.AttributeEnd = __rilc_AttributeEnd;
    JumpLC.Basis = __rilc_Basis;
    JumpLC.Sides = __rilc_Sides;
    JumpLC.Orientation = __rilc_Orientation;
    JumpLC.ReverseOrientation = __rilc_ReverseOrientation;
    JumpLC.Identity = __rilc_Identity;
    JumpLC.Transform = __rilc_Transform;
    JumpLC.ConcatTransform = __rilc_ConcatTransform;
    JumpLC.Perspective = __rilc_Perspective;
    JumpLC.Translate = __rilc_Translate;
    JumpLC.Rotate = __rilc_Rotate;
    JumpLC.Scale = __rilc_Scale;
    JumpLC.TransformBegin = __rilc_TransformBegin;
    JumpLC.TransformEnd = __rilc_TransformEnd;
    JumpLC.SurfaceV = __rilc_SurfaceV;
    JumpLC.Illuminate = __rilc_Illuminate;
}

int XRes = 256;
int YRes = 256;

extern void init_lightshaders(void);
extern void init_surfaceshaders(void);
extern void init_attributes(void);
extern void init_tokentables(void);
extern void init_options(RiOptions *opt);
extern void __init_luts(void);
extern void __init_parser(void);

RtVoid RiBegin(RtToken name)
{
    float zero[4] = { 0.,0.,0.,0. };

    RenderState |= STATE_BEGIN;

    if( name!=RI_NULL ) {
      fprintf(stderr,"unsupported begin implementation\n");
    }

    /* Create the rendering window and widgets */
    if( getenv("XRES") )
      XRes = atoi(getenv("XRES"));
    if( getenv("YRES") )
      YRes = atoi(getenv("YRES"));
    xprefsize(XRes,YRes);
    xprefposition(100,100+XRes,100,100+YRes);
#if 0
    if( getenv("CHECKSUM")||getenv("IMAGE") ) {
        xprefposition(100,100+XRes,100,100+YRes);
    }
#endif
    /* xdoublebuffer(); */
    xwinopen("The RenderMan Interface");
    /* reshape viewport */
    xpolldevices(NULL);

    glClearColor(.0,.0,.0,0.);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(.0,.0,.0,0.);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    glEnable(GL_MULTISAMPLE_SGIS);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,zero);
    glLightModelf(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
    glFragmentLightModelfvSGIX(GL_FRAGMENT_LIGHT_MODEL_AMBIENT_SGIX,zero);

glFragmentLightModelfSGIX(GL_FRAGMENT_LIGHT_MODEL_LOCAL_VIEWER_SGIX,GL_TRUE);

    /* default is two sided lighting and left handed orientation */
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE);
    glFragmentLightModelfSGIX(GL_FRAGMENT_LIGHT_MODEL_TWO_SIDE_SGIX,GL_TRUE);
    glFrontFace(GL_CW);
    glCullFace(GL_BACK);
    glDepthFunc(GL_LEQUAL);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* must do before init_attributes */
    __ri_initmemmgr();

    init_jumptables();
    JumpCur = &JumpIm;
    init_tokentables();
    init_lightshaders();
    init_surfaceshaders();
    init_options(CurOptions);
    init_attributes();

    /* initialize information needed for parsing and executing shaders.
       the lut initialization must happen before the parser because
       the parser depends on tokens created in the luts call. */
    __init_luts();
    __init_parser();

    /* CurLights = globallight; */
}

#define CRCMASK 0x04c11db7

static unsigned int crcinit(unsigned int crc)
{
    int i;
    unsigned int ans = crc;
    for (i=0; i < 8; i++) {
        if (ans & 0x80000000) {
            ans = (ans << 1) ^ CRCMASK;
        } else {
            ans <<= 1;
        }
    }
    return ans;
}

static unsigned int crctab[256];

static unsigned int crcgen(unsigned char *bufp, int len)
{
    unsigned int i, cword = ~0;
    static int crcinited = 0;

    if (!crcinited) {
        /*
         * Initialize a lookup table for the 8 most significant bits of the
         * cumulative remainder.  This way we can do the division 8 bits at
         * a time, instead of 1 at a time.
         */
        for (i=0; i < 256; i++) {
            crctab[i] = crcinit(i << 24);
        }
        crcinited = 1;
    }
    for (i=0; i < len; i++) {
      if( (i%4)!=3 ) /* ignore alpha channel */
        cword = crctab[ bufp[i] ^ (cword >> 24) ] ^ (cword << 8);
    }
    return cword;
}

RtVoid RiEnd(void)
{
    /* delete_lights(globallight); */

    RenderState &= ~STATE_BEGIN;

    if( getenv("CHECKSUM")||getenv("IMAGE") ) {
        if( getenv("CHECKSUM") ) {
          unsigned char *im;

          im = (unsigned char *)malloc(4*CurOptions->hres*CurOptions-
>vres*sizeof(unsigned char));
            glReadPixels(0,YRes-CurOptions->vres,CurOptions->hres,
                                CurOptions->vres,GL_RGBA,GL_UNSIGNED_BYTE,im);

          printf("%s:  0x%08x\n",CurOptions-
>displayname,crcgen(im,4*CurOptions->hres*CurOptions->vres));
          free(im);
      }
        if( getenv("IMAGE") ) {
            extern void writergbaimage(char *n, int xsize, int ysize,
                                    float *image);
          float *fim;
          fim = (float *)malloc(4*CurOptions->hres*CurOptions-
>vres*sizeof(float));

            glReadPixels(0,YRes-CurOptions->vres,CurOptions->hres,
                                CurOptions->vres,GL_RGBA,GL_FLOAT,fim);
          writergbaimage("image.rgb",CurOptions->hres,CurOptions->vres,fim);
          free(fim);
      }
    } else {
        xwaitescape();
    }
}

/*ARGSUSED*/
RtVoid RiFrameBegin(RtInt number)
{
    extern void push_options(void);

    RenderState |= STATE_FRAME;

    push_options();
    __riim_AttributeBegin();
    /* XXX push tranformation matrix for camera */
    glPushMatrix();

    /* CurLights = framelight; */
}

RtVoid RiFrameEnd(void)
{
    extern void pop_options(void);

    RenderState &= ~STATE_FRAME;

    /* delete_lights(framelight); */

    /* XXX pop tranformation matrix for camera? */
    glPopMatrix();
    __riim_AttributeEnd();
    pop_options();
}




static void set_camera(void)
{
    float xmin, xmax, ymin, ymax;
    int x, y, dx, dy;

    /* from riformat:  upper-left corner of screen */
    glViewport(0,YRes-CurOptions->vres,CurOptions->hres,CurOptions->vres);

    /* crop relative to upper-left corner of screen */
    x = (int)((float)CurOptions->hres*CurOptions->cropwindow[0]);
    dx = (int)((float)CurOptions->hres*
                (CurOptions->cropwindow[1]-CurOptions->cropwindow[0]));
    y = (int)((float)CurOptions->vres*CurOptions->cropwindow[2]);
    dy = (int)((float)CurOptions->vres*
                (CurOptions->cropwindow[3]-CurOptions->cropwindow[2]));
    glEnable(GL_SCISSOR_TEST);
    glScissor(x,YRes-CurOptions->vres+y,dx,dy);

    /* screen window */
    xmin = CurOptions->screenwindow[0];
    xmax = CurOptions->screenwindow[1];
    ymin = CurOptions->screenwindow[2];
    ymax = CurOptions->screenwindow[3];

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    /* combination of screen transformation and projection */
    if( CurOptions->projection==RI_ORTHOGRAPHIC ) {
        glOrtho(xmin,xmax,ymin,ymax,-100.,100.);
    } else {
#if 1
      if( getenv("JITTERX") ) {
          glTranslatef(atof(getenv("JITTERX"))/(float)CurOptions->hres,0,0);
      }
      if( getenv("JITTERY") ) {
          glTranslatef(0,atof(getenv("JITTERY"))/(float)CurOptions->vres,0);
      }
        glTranslatef(-(xmax+xmin)/(xmax-xmin),-(ymax+ymin)/(ymax-ymin),0.);
        glScalef(2./(xmax-xmin),2./(ymax-ymin),1.);
        gluPerspective(CurOptions->camera_fov,1.,
                  CurOptions->clip[0],CurOptions->clip[1]);
#else
        glFrustum(CurOptions->clip[0]*xmin,CurOptions->clip[0]*xmax,
                  CurOptions->clip[0]*ymin,CurOptions->clip[0]*ymax,
                  CurOptions->clip[0],CurOptions->clip[1]);
#endif
    }
    /* looking to positive z */
    glScalef(1.,1.,-1.);

    glMatrixMode(GL_MODELVIEW);
}

/* here is where the action begins.  we have simply collected all of the
   option information up to this point.  now the options can not change
   until worldend, when the image is completed.  we must apply all of
   the options that have been specified.  the most obvious is the
   camera transformation.  but things like shutter time and filtering
   are employed here.  */

RtVoid RiWorldBegin(void)
{
    /* xwinposition(100,100+XRes,100,100+YRes); */

    RenderState |= STATE_WORLD;
    glClearColor(.0,.0,.0,0.);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(.0,.0,.0,0.);

    set_camera();
    glGetFloatv(GL_MODELVIEW_MATRIX,(GLfloat *)CurOptions->worldtocamera);
    glMatrixMode(GL_PROJECTION);
    /* glMultMatrixf((GLfloat *)CurOptions->worldtocamera); */

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glPushMatrix();

    /* set the raster position for copies */
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, 1, 0, 1, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glRasterPos2i(0, 0);

    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);

    __riim_AttributeBegin();

    JumpCur = &JumpLC;

    /* CurLights = worldlight; */
}

char *__cur_onoff;
int num_passes = 0;

RtVoid RiWorldEnd(void)
{
    extern void __lt_run_lightshaders(Node *);

    Node *node = RiScene;

    RenderState &= ~STATE_WORLD;

    __lt_run_lightshaders(RiScene);

    while( node!=NULL ) {
      __cur_onoff = node->light;
      node->shader(&node->surf,&node->att,node->dlist);
      node = node->next;
    }
    /* fprintf(stderr,"TOTAL PASSES:  %d\n",num_passes); */

    JumpCur = &JumpIm;
    delete_lights(worldlight);

    glPopMatrix();
    __riim_AttributeEnd();

    /* return to the surface active before worldbegin */
    /* XXX must be fixed */

    RiScene->next = NULL;
    RiScene->last = RiScene;
}
```